CS/INFO 3300 / INFO 5100
Homework 1
Due 11:59pm Monday 2/5

Goals: Review Javascript, particularly functional programming. Functions can return functions, and can take functions as arguments. Practice simulating from random processes. Practice iteration with for-loops and updating web elements from Javascript. Experiment with a strange set of dice created by Bradley Efron, whom we will meet again.

Your work should be in the form of an HTML file called `index.html` with one `<p>` element per problem. Wrap any Javascript code for each problem in a `<script>` element. For example:

```
<p id="p0">Problem 0: We use the var statement to declare a
variable and set it to a value.
<script>
var x = 100;
</script>
</p>
```

(Note that functions and variables that you define in one `<script>` element are available in all subsequent `<script>` elements. You do not need to repeat code.)

Create a zip archive containing this file and upload it to CMS.

For each problem, you may use your work from the previous problem (though you don't have to). Do not use any other non-standard libraries or resources (`Math.random()` is ok, jQuery is not).

1. [Die simulator] Write a function called `roll` that returns a random value from `[1, 2, 3, 4, 5, 6]`, with equal (*uniform*) probability. Run this function 10 times and display the results in the `p` tag for this problem. The function `join` on an array may be useful when displaying results. (20 pts)

2. [Weird die simulator] Write a function called `diceFactory` that takes one argument, an array called `values`, and returns a function. This returned function should work the same way as your `roll()` function, but return a uniform sample from the `values` array rather than digits from 1-6. Create a function `lunch` by

calling `diceFactory(["pizza", "burrito", "sushi"]).` Run the `lunch` function 10 times and display the results in the `p` tag for this problem. (20 pts)

3. [Non-transitive dice generator] I recently got some fun conference swag: Bradley Efron's non-transitive dice. These are four six-sided dice that have the following face values:

```
     [1, 1, 1, 5, 5, 5] [0, 0, 4, 4, 4, 4] [3, 3, 3, 3, 3, 3]
[2, 2, 2, 2, 6, 6]
```

We'll call these [`green`, `blue`, `yellow`, and `orange]` (visit my office to see why). Write a new dice-rolling function generator function called `nonTransitiveFactory` that returns a function that simulates one of these dice. The generator function should take one argument, a number from {0, 1, 2, 3}, that specifies which die to return. Use the generator function to create a dice-rolling function `greenDie` that samples from the `green` array. Use a `for` loop to run this function 10 times and display the results in the `p` tag for this problem. (20 pts)

4. [Rolling many times] Write a function called `rollNTimes` that takes three arguments, two dice-rolling functions (*i.e.* functions returned by your `nonTransitiveFactory` function) and a number `n`. It should "roll" the two dice `n` times and return the number of times the first function returned a value greater than the second function (ties don't count). Run this function 10 times with `green` and `blue` with `n=100`, and display the results in the `p` tag for this problem. (20 pts)

5. [All pairs comparison] Now create a 4x4 table where each cell has a distinct `id` attribute (you can write it out explicitly or generate it automatically). Set the background color for each cell to be the color associated with the *row* of the table. For each possible pair of dice `i, j`, use the `innerText` property to set the value in the appropriate table cell to the result you get by calling `rollNTimes` with the appropriate functions, and with `n=1000`. It's ok to compare a die to itself and to display both symmetrical pairs. In the `p` tag for this problem describe why these are called non-transitive dice. (20 pts)